chapter

7 Let the Data Drive

In This Chapter

- Good Things Come in Free Packages
- New Functions
- Installing MySQL
- Setting up a Simple Database
- Basic SQL Queries
- Putting Content into a Database Using PHP
- Getting Content from a Database Using PHP

◆ Good Things Come in Free Packages

Data-driven Web sites—you've heard the term thrown about the local Starbucks® a million times, spoken by men and women with really nice shoes and really small laptops. You've heard about the power of Microsoft's® Active Server Pages™, and of the awe and might of Allaire's® Cold Fusion. But you think to yourself, *We don't have any more money in the budget; there's no way we can afford one of those commercial databases or application servers. Besides that, who's going to develop it? The developer alone is out of our price range.*

Luckily, you can do it yourself using tools that are easier to use and can produce the same results as expensive commercial

applications can give you. In this chapter you'll be introduced to the free[1] MySQL database and shown how to use it with PHP to create your own data-driven Web sites.

Integrating your Web site with a database opens up worlds of possible features that you can add to give your users a more interactive and robust experience while visiting your site. It also makes it easier for you, as a developer, to create a site by using modularized pieces of information stored in a database.

If you did the earlier projects (and I know you did—you wouldn't skip ahead would you?), you can easily use PHP to connect to a database and join in on the data-driven Web site frenzy.

◆ New Functions

printf()

`printf()` is basically the same as `print()`, only it gives you a little more precision as to how you want to format the display of your variables. In the following example, the `%s` is replaced by the variables at the end of the statement. The `%s` tells the `printf` function to format the variables as strings, as opposed to, say, an integer or binary number.

```
printf("<TR><TD>%s</TD><TD>%s</TD></TR>\n", $row["col1"],
$row["col2"]);
```

while

The `while` statement is a simple looping mechanism that allows you to evaluate a set of data until it comes across a value that doesn't meet the required criteria. The loop is exited once the evaluation against the criteria is proven false. This function is useful if you don't know the size of the data set you need to evaluate—for instance, some data retrieved from a database.

```
while($a == $b) {

    print("<P> $a is equal to $b \n");

}
```

---

1. See the License Information that comes with MySQL for full details. A MySQL license must be purchased for Windows™ servers after a 30-day trial period. There are some additional license restrictions for commercial repackaging of MySQL.

mysql_connect

The `mysql_connect` function establishes a connection with a MySQL Server. It takes as its arguments the MySQL server location, your MySQL login, and your MySQL password.

This function is usually assigned to a variable that is then used in the `mysql_select_db` function.

```
$server_connection  = mysql_connect("localhost", "chris",
"password1");
```

NOTE:
The password argument is necessary only if you have set a MySQL password for yourself. If you don't have a password, and don't need one to connect to the server, you can leave the password argument out of the function.

mysql_select_db

The `mysql_select_db` function selects a particular MySQL database on a MySQL server. It takes as its arguments the database name and the MySQL server connection. You must use this function to select a database on the server before you can send queries.

```
mysql_select_db("news", "$server_connection");
```

mysql_query

The `mysql_query` function sends SQL queries to a database on a MySQL server. It takes as its argument an SQL query. If you are connecting to more than one MySQL server in a script, then you can specify which connection you want to query by adding a server connection to the arguments.

```
$result = mysql_query("SELECT * FROM news");

$result = mysql_query("SELECT * FROM news",
"$server_connection");
```

mysql_fetch_array

The `mysql_fetch_array` function takes the first row of the result of a MySQL query and puts it into an array. Each call to this function gets the next row of the data from the result of the query

until no rows are left. Either the column number or the column name can reference the array.

```
$myrow = mysql_fetch_array($result);
```

◆ Installing MySQL

Installing MySQL is fairly straightforward for both Windows and Linux. You can get the latest versions of MySQL from *http://www.mysql.com/download.html*. From there, click on the appropriate download link for your operating system.

If you are using Windows 95/98 or NT, download the latest shareware version. You'll need to fill out a short registration questionnaire before you can download the setup program.

Once you have downloaded the shareware version, extract the zip file and execute the *setup.exe* program. The setup program should install the MySQL files in C:\mysql.

Then all you need to do is start the MySQL server daemon by executing the command *C:\mysql\mysqld –install.* Subsequently, you can start the MySQL server daemon by issuing the command *C:\mysql\mysqld.* Remember that the MySQL server daemon must be running if you want to connect to your databases using PHP.

If you are using a flavor of Linux that supports Red Hat's RPM format, I strongly recommend downloading one of the RPM binary versions of the latest stable release of MySQL. The RPM versions install all the necessary startup scripts, and generally take a lot of the headaches out of compiling your own binary version from the source code.

To install the RPM version of MySQL just issue the command *rpm –ivh <MYSQL-RPM>.rpm,* where <MYSQL-RPM> is the name of the MySQL RPM package that you downloaded. This package automatically installs the startup scripts and configures the default databases.

As always, be sure to read the installation documentation that is available on the MySQL Web site for more detailed information.

◆ Setting up PHP to Work with MySQL

After you have installed MySQL, you'll need to make a few minor adjustments to your PHP installation before the two programs will play nice with each other. For Windows users, this means

you need to make one small change to your .ini file. For Linux users, this means that you'll have to recompile PHP once again.

If you are using the Windows version of PHP, you need to edit the *php3.ini* file (or *php.ini* if you are using PHP4) in C:\windows or C:\WINNT, depending on your operating system.

To enable PHP to work with MySQL, you need to uncomment the line that says "extension=php3_mysql.dll." To do this you need to delete the semicolon from in front of that line. The line should look like the one below.

```
;Windows Extensions
extension=php3_mysql.dll
```

Now save the file and restart your Apache server, and you are all set to go.

If you are using Linux, you need to recompile your version of PHP. While this may take a few more minutes to set up than if you were using Windows, you'll once again have the satisfaction of turning a bunch of source code into a living, breathing program. After all, that's why you are a Linux user!

When you recompile, just follow the same steps you used in Chapter 1, only add the flag—`with-mysql` when you issue the `configure` command. This tells the PHP compile program to include the extra functions you'll need when connecting to the database.

After that, you'll need to stop and restart your Apache server for the changes to take effect.

◆ SQL Queries

SQL (Structured Query Language) is a powerful, robust, and sometimes complicated query language that lets you manipulate information in databases. Luckily, you can get by at first with only the most basic knowledge of the SQL language to use it effectively in PHP. Table 7.1 lists some of the most frequently used commands. In the usage examples, `news` is the name of the table, and `news_id` is the name of a column in the table.

In general, it's good SQL programming practice to capitalize the SQL language keywords, such as SELECT, WHERE, and UPDATE. This makes your SQL queries easier to read, and easier for other programmers to look at your code and understand what is going on.

TABLE 7.1 SQL Basics

Keyword	Definition and Usage
SELECT	Used to select data from the database. The data that you select is put into a table. **Usage** `SELECT * FROM news WHERE news_id > '1'`
INSERT	Used to insert new information into the database. **Usage** `INSERT INTO news VALUES(NULL,'1','10-22-2000','Chris')`
DELETE	Used to delete a row of data from a database. **Usage** `DELETE FROM news WHERE news_id = '5'`
UPDATE	Used to modify data in the database. **Usage** `UPDATE news SET urgent = 'YES' WHERE news_id = '5'`

◆ Setting up a Simple Database

First you need to create a database to store your data. Make certain you have your MySQL daemon running. Replace `DBNAME` with whatever name you want to name your database. If you are using Windows, you need to execute any MySQL commands from C:\mysql\bin. For Linux, you can issue the commands from any shell prompt. To create your database, issue the following command:

```
prompt>mysqladmin create DBNAME
```

To set up the tables in your new database, you need to login to your MySQL server. This is done by using the MySQL command-line interface.

1. At the prompt, issue the following command:

   ```
   prompt> mysql –u root
   ```

   This command logs you into the server. From here you can create tables, delete (drop) tables, and modify the data in your tables. But first, you must specify which

database on the server you want to use, since the MySQL server is capable of hosting multiple databases.

2. At the MySQL prompt, enter the following command and be sure to replace `DBNAME` with your database's name.

```
mysql> use DBNAME;
```

The MySQL server responds with:

```
Database changed
```

You've now selected your database. Any of the basic SQL queries you enter are directed to this database.

3. Now you can create the table you'll be using for the upcoming projects. Enter the following commands at the MySQL prompt. Hit **Enter** after each line. MySQL doesn't try to interpret the commands until it sees a semicolon (;), so the command itself isn't really executed on the server until you enter the last line.

```
mysql> CREATE TABLE news (
    -> news_id INT NOT NULL AUTO_INCREMENT,
    -> heading VARCHAR(48),
    -> body TEXT,
    -> date DATE,
    -> author_name VARCHAR(48),
    -> author_email VARCHAR(48),
    -> PRIMARY KEY(news_id));
```

If the server gives you a big **ERROR** and spits out a bunch of garbage at you, just try again from the top. You need to enter each line in the sequence from the beginning, exactly as shown.

NOTE:
The → prompt you see after entering a line in the MySQL server is telling you that it's waiting for more input before it does anything.

The server responds with:

```
Query OK, 0 rows affected (0.00 sec)
```

Congratulations! You've just created your first table in MySQL.

◆ Basic SQL Queries

Now that you've set up your first table, you can practice some of the basic SQL queries using the command-line interface of the MySQL server. These exercises will help you get comfortable with basic SQL queries.

First, let's talk about the structure of your new table before we start entering data into it. If you want to see the structure of a particular table, you can ask MySQL to describe it by simply issuing the command `describe`.

Type:

```
mysql> describe news;
```

and the server displays the description of your `news` table

```
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| news_id      | int(11)     |      | PRI | 0       | auto_increment |
| heading      | varchar(48) | YES  |     | NULL    |                |
| body         | text        | YES  |     | NULL    |                |
| date         | date        | YES  |     | NULL    |                |
| author_name  | varchar(48) | YES  |     | NULL    |                |
| author_email | varchar(48) | YES  |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

The table that MySQL returns from the `describe` command gives you all the relevant details of your table.

Here is a brief explanation of the output from the `describe` command:

- Each of the `Field`s holds your data.
- The `Type` of field is what kind of data that particular field can hold.
- If the field has `YES` in its `Null` column, then it can be empty for any row on the table.
- If the field has `PRI` in the `Key` column, then that field acts as an index of sorts for the table. No two values can be the same in any row of a table that is the primary key.
- The `Default` column describes what values, by default, are entered into the field.
- Finally, the `Extra` column describes any extra attributes a row might have, such as `auto_increment`, which increments your primary key by 1 every time you add a new row into the table.

Inserting Data

Now, let's put some data into your new table. Type in the following code at the MySQL prompt. Remember to hit enter at the end of each line.

```
mysql> INSERT INTO news
         -> VALUES(NULL,'A Heading','The Body',
         -> '08-16-2000','Chris C.',
         -> 'chris@domain.com');
```

The server responds with:

```
Query OK, 1 row affected (0.00 sec)
```

Now let's query the database and see how it looks in the table by issuing a `select` statement. The * operator works just like a standard UNIX wildcard.

Type the following at the MySQL prompt:

```
mysql> select * from news;
```

The server responds with:

```
+---------+-----------+----------+------------+-------------+------------------+
| news_id | heading   | body     | date       | author_name | author_email     |
+---------+-----------+----------+------------+-------------+------------------+
|       1 | A Heading | The Body | 0000-00-00 | Chris C.    | chris@domain.com |
+---------+-----------+----------+------------+-------------+------------------+
1 row in set (0.06 sec)
```

Modifying Data

Now what if you entered something incorrectly and you want to modify it? Let's look at the above output. Notice anything strange? The date is listed as 0000-00-00. And no, that isn't some kind of Y2K bug. It's because we didn't enter the date into the table using the correct format. We tried to enter it as *08-16-2000* when we should have entered *2000-08-16*. The date type of data in MySQL won't accept a value that doesn't fit the required parameters. The required parameters state that the date must be a four-digit year, followed by a dash, followed by a two-digit month, followed by a dash, followed by a two-digit day.

To correct the error, you can issue the `update` command. Type the following at the MySQL prompt:

```
mysql> update news set date='2000-08-16' where news_id='1';
```

The server responds:

```
Query OK, 1 row affected (0.00 sec)
```

Just to make sure everything is correct, let's issue the `select` command again. Type the following at the MySQL prompt:

```
mysql> select * from news;
```

The server responds:

```
+---------+-----------+----------+------------+-------------+------------------+
| news_id | heading   | body     | date       | author_name | author_email     |
+---------+-----------+----------+------------+-------------+------------------+
|       1 | A Heading | The Body | 2000-08-16 | Chris C.    | chris@domain.com |
+---------+-----------+----------+------------+-------------+------------------+
1 row in set (0.06 sec)
```

And as you can see from the output, the date is now entered correctly.

Deleting Data

Now that you have learned how to insert, retrieve, and modify data in your database, the last basic command you need to know is the `delete` command.

To delete a row in your table, type this command at the MySQL prompt:

```
mysql> delete from news where news_id='1';
```

The server responds:

```
Query OK, 1 row affected (0.00 sec)
```

Now do a select statement and see what the table looks like.

```
mysql> select * from news;
```

The server responds with:

```
Empty set (0.00 sec)
```

MySQL returns "Empty set" because the criteria you used for the `select` statement (the * operator) returned no values, so this table is empty. If you had used some form of criteria, such as `select * from news where news_id='5'`, an empty set might also be returned, but there could still be data in the table. For example, the table could still contain data where `news_id` equals 2, or 3, or 4. But since you asked for everything in the table, and nothing was returned, you know the table is empty.

The `delete` command simply deletes a row (or rows) that meets the criteria. In this case you deleted the row from your table that had a `news_id` of 1. You could have also chosen to delete rows that had an `author_name` of Chris C., which could delete multiple rows. If the data had hundreds of rows, we could delete them all by using a command such as `delete from news where news_id > '0';`.

◆ Putting Content into Your Database with PHP

Now that you have had a little experience using SQL queries from the MySQL command line, let's try some queries using PHP. Using PHP for the task is less cumbersome, more flexible, and above all, it can easily be done using a Web browser.

The logic behind PHP and database interaction is simple.

- Connect to the database server and login.
- Choose the database to use.
- Send SQL queries to the server to add, delete, and modify data.

You don't even have to worry about closing the connection to the server because PHP does it for you.

Script 7-1 shows you just how easy it is to put data into a database. What PHP allows you to do in a few short, simple lines is amazing. This script uses the `news` MySQL table that you created earlier. If you haven't done so, you need to create it now. Out of the thirty lines of code in the script, there are only about ten that are PHP-specific. The rest is plain HTML. Examples are shown in Figures 7–1 and 7–2.

Script 7-1
data_in.php3

```
1. <html>
2. <head>
3. <title>Putting Data in the Database</title>
4. </head>
5. <body bgcolor="#FFFFFF">
6. <?php
7. /* This program enters news items into a database */
8. if(isset($submit)):
9. $db = mysql_connect("localhost", "root");
```

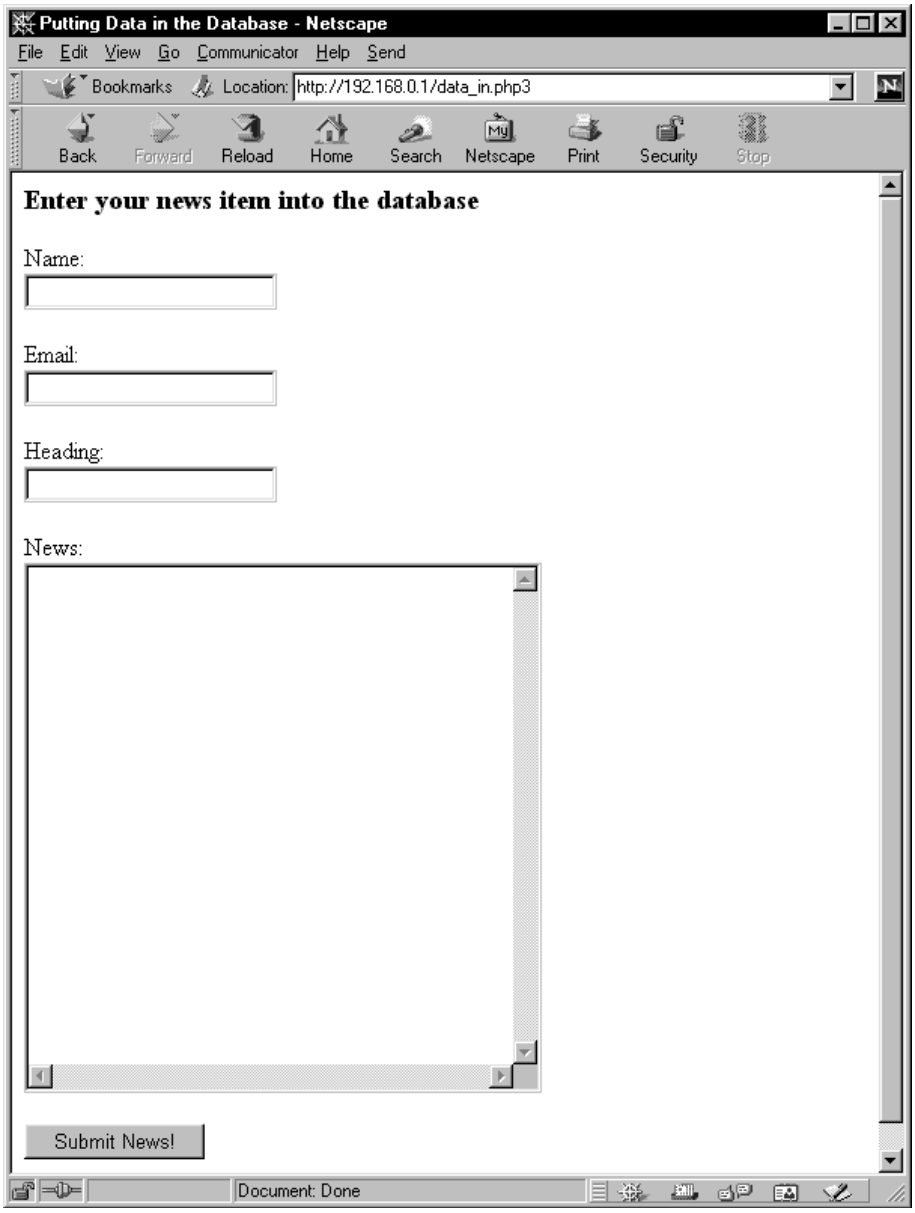

FIGURE 7–1 Initial news entry form from Script 7-1

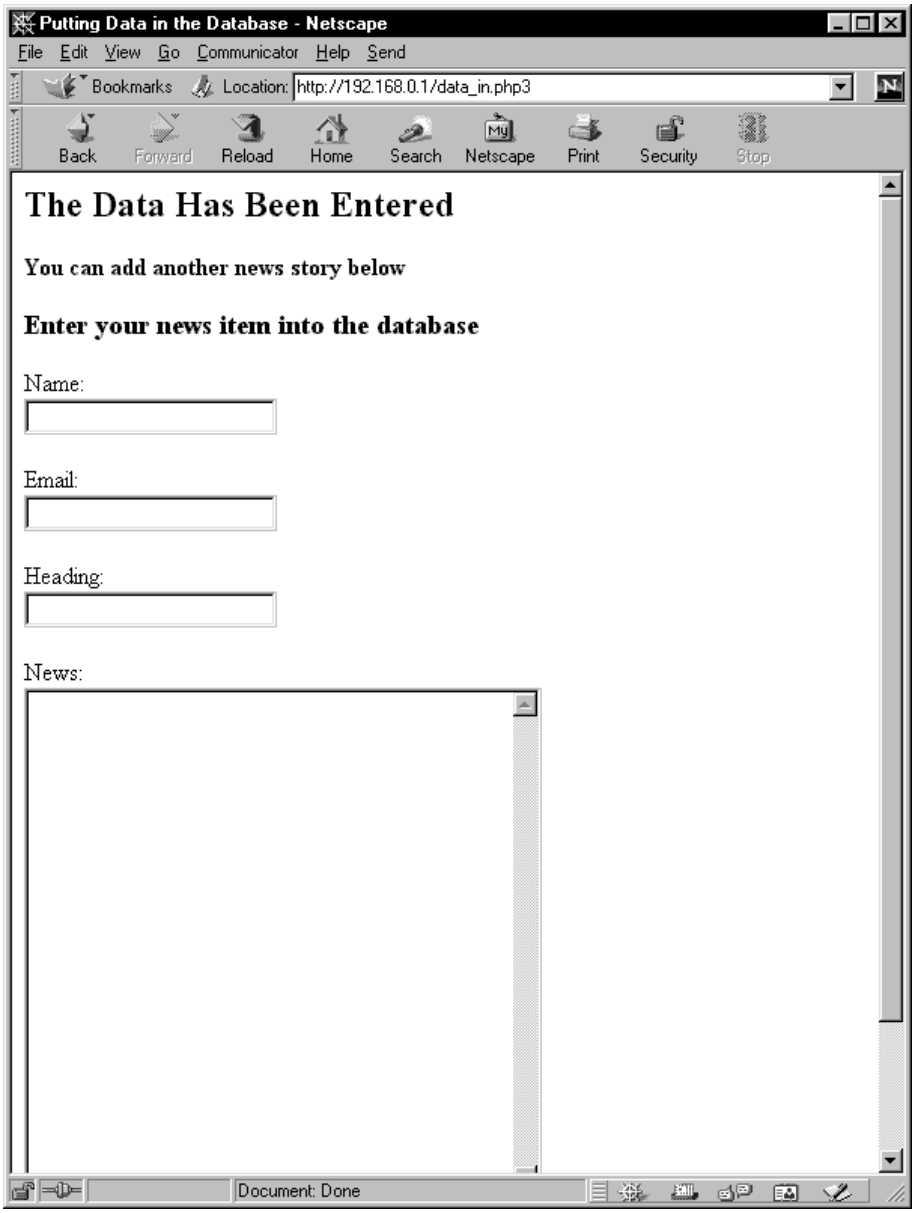

FIGURE 7–2 Result of the form submission from Script 7-1

```
10. mysql_select_db("php3", $db);
11. $date = date("Y-m-d");
12. $sql = "INSERT INTO news
13. VALUES(NULL,'$heading','$body','$date',
    '$auth','$auth_email')";
14. mysql_query($sql);
15. print("<h2>The Data Has Been Entered</h2>\n");
16. print("<b>You can add another news story below</b><hr>\n");
17. endif;
18. ?>
19. <p><h3>Enter your news item into the database</h3>
20. <form action="data_in.php3" method="post">
21. Name:<br><input type="text" name="auth"><p>
22. Email:<br> <input type="text" name="auth_email"><p>
23. Heading:<br><input type="text" name="heading"><p>
24. News:<br>
25. <textarea cols=40 rows=20 name="body" wrap="virtual">
26. </textarea><p>
27. <input type="submit" name="submit" value="Submit News!">
28. </form>
29. </body>
30. </html>
```

HOW THE SCRIPT WORKS

- Line 8 checks to see if the Submit button has been pressed. If it has, it executes the code. If not, it skips to the `endif` part of the script. In this script, if the button isn't pressed, the PHP code is almost completely ignored. This makes for faster initial loading of the page.
- Line 9 is a variable is assigned to the `mysql_connect` function. This variable is used in the next line to actually establish the connection. The arguments for the function tell the script to connect to the MySQL server running on `localhost`, and to login to it with a username of `root`. This particular server doesn't require a password, so it has not been included as an argument to the function.
- Line 10 calls the function `mysql_select_db`, which selects the database you want to use, and it also initiates the connection to the server by calling the value of the `$db` variable.
- Line 12 is the SQL statement. In this case the statement is assigned to a variable so that the `mysql_query` is easier to read.
- Line 14, the `mysql_query` function, sends the SQL statement to the MySQL server. The SQL statement that is being sent

tells the MySQL server to enter the values it received from the form into the database table. Here the script also prints out a message letting the user know that his or her data was entered.

- Line 19 returns to plain old HTML. The beauty of this script is that there isn't a lot of overhead. It lets PHP do the hard stuff, and lets HTML do the rest. We could have created a function to print the form to the browser, but that would just put more work on the server because it would have to process more PHP directives. With this script, we keep things simple, fast, and efficient.

The script also repeats itself so that multiple news items can be entered in one session. The HTML form gets printed out every time the script is run, and every time the user hits Submit more data is entered into the database.

Now, of course, this is an ultra-simple example. You would want to include the error-checking statements you learned earlier in the book, and you would also want to spice it up a little and make it easier for the eyes of your users by using a simple style sheet.

◆ Getting Content out of Your Database with PHP

OK, so you've got your PHP scripts set up to put as much data as you want into the database. Now what? Well, you could always go back to that not-so-user-friendly command-line interface that MySQL offers you and do `select` statements all day and night to see what's in there, or you could do it the easy way with PHP.

The basic PHP/MySQL logic is still the same to get data out of the database. You connect to the database server and login; choose the database to use; send SQL queries to the server to add, delete, and modify data—*voila.*

This script also introduces you to the `while` statement. You'll use the `while` statement because it's not always definite how much data your SQL queries will return. You may get one row of data from a query, or you might get fifty. The `while` statement lets you go through all the data returned from your query, then stops when the data ends. Examples of this are shown in Figures 7–3, 7–4, and 7–5.

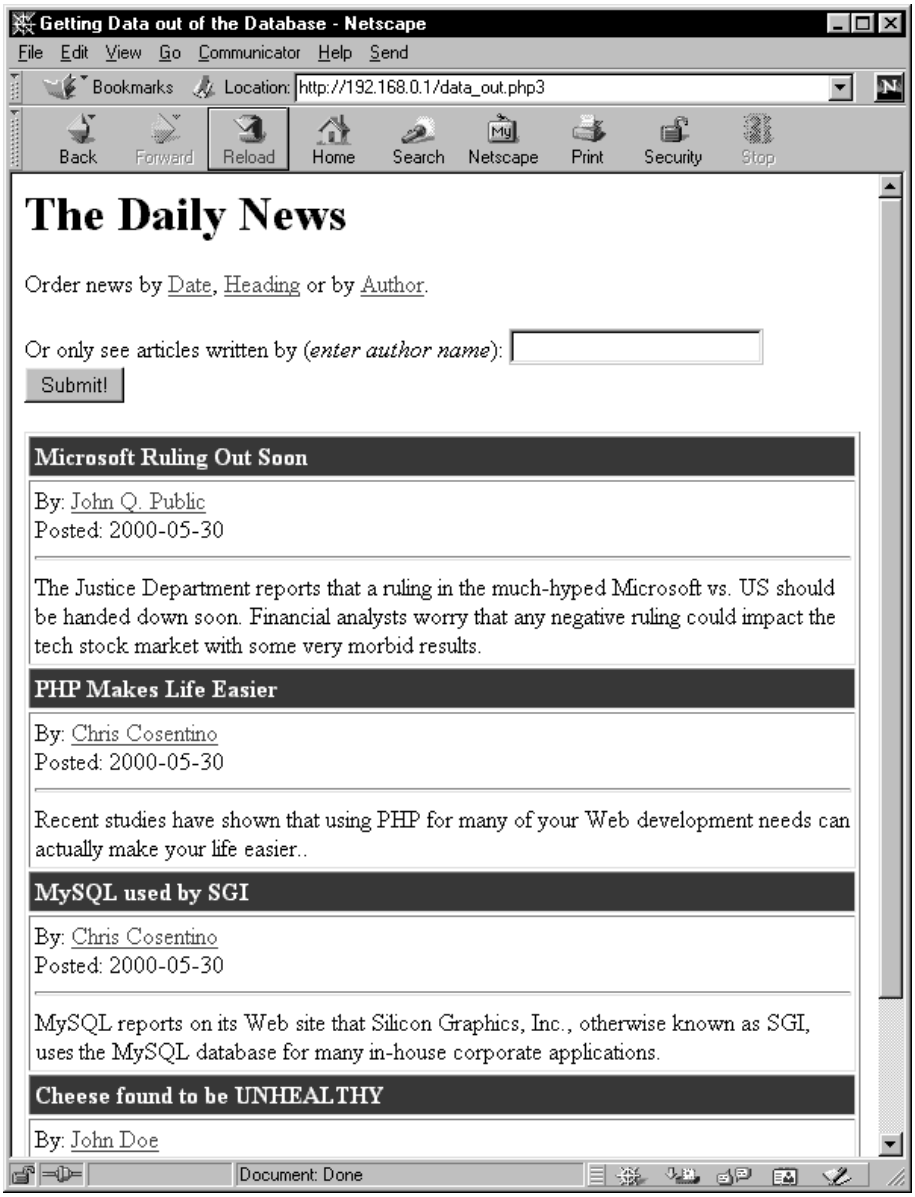

FIGURE 7–3 Initial load of Script 7-2

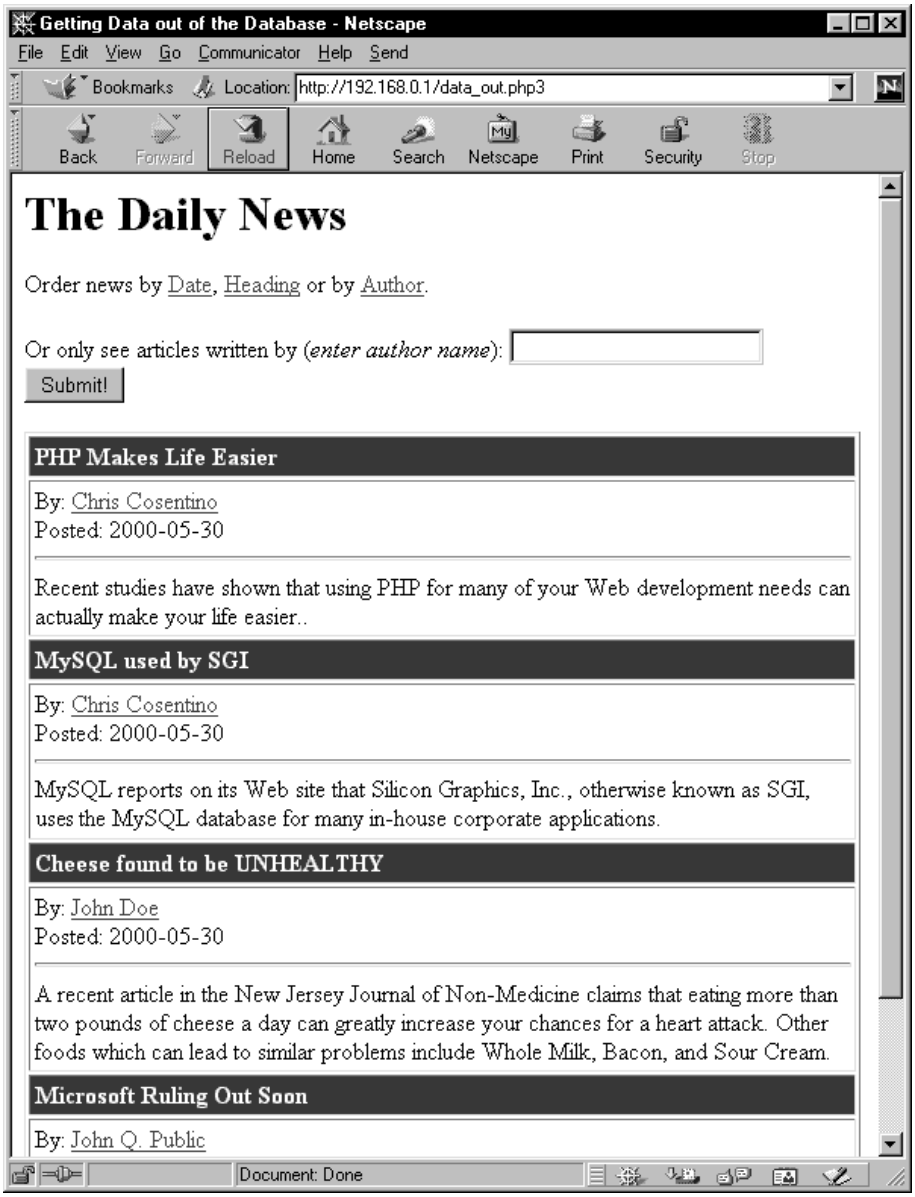

FIGURE 7–4 Script 7-2 after clicking on the Order by Author link

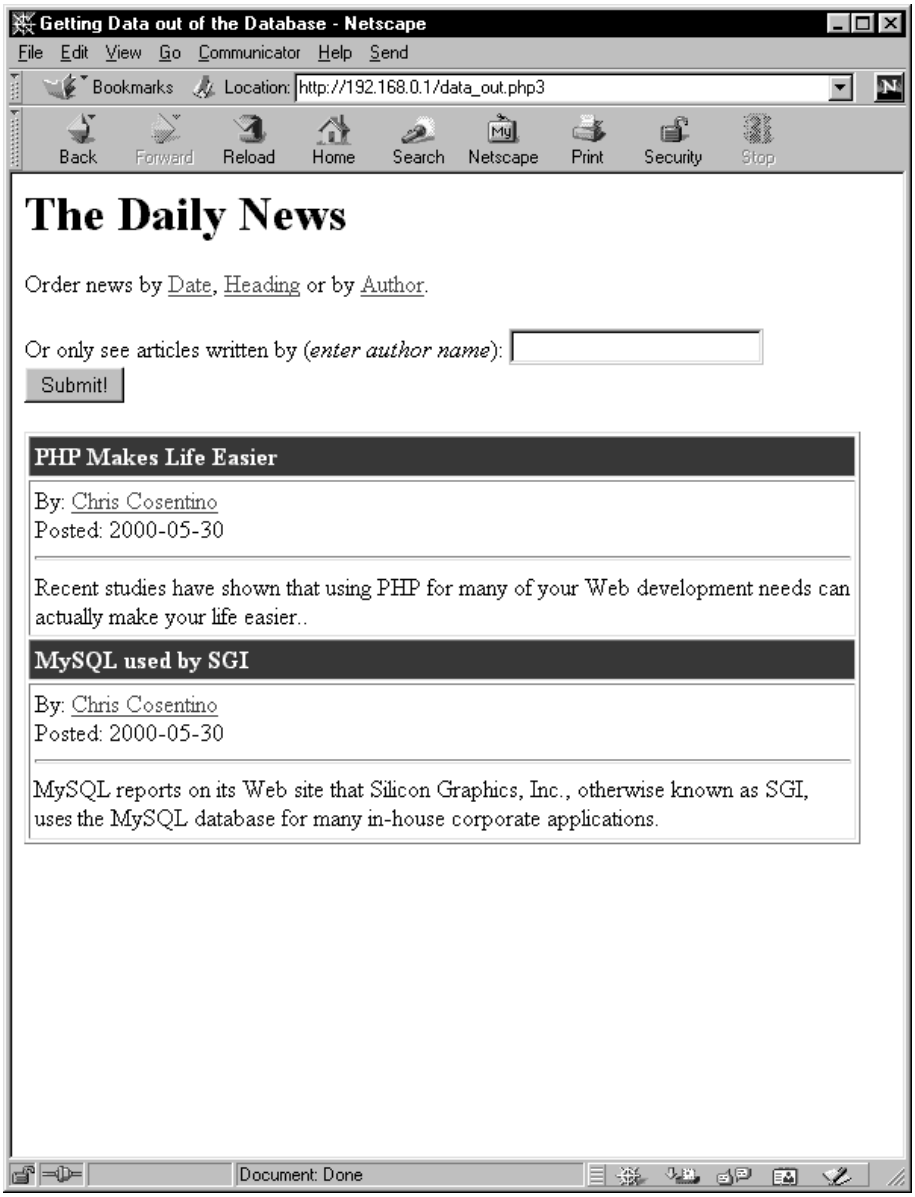

FIGURE 7–5 Script 7-2 displaying articles by author

Since the basic concepts are the same, we'll spice up Script 7-2, with a few extra features to enhance the user's viewing experience. We'll give the user the choice of how he or she wants the information ordered, and we'll also give the user the choice of which author's articles he or she wants to view.

Script 7-2
data_out.php3

```
 1. <html>
 2. <head>
 3. <title>Getting Data out of the Database</title>
 4. </head>
 5. <body bgcolor="#FFFFFF">
 6. <h1>The Daily News</h1>
 7. Order news by
 8. <a href="data_out.php3?orderby=date">Date</a>,
 9. <a href="data_out.php3?orderby=heading">Heading</a> or by
10. <a href="data_out.php3?orderby=author">Author</a>.
11. <p>
12. <form action="data_out.php3" method="POST">
13. Or only see articles written by (<i>enter author name</i>):
14. <input type="text" name="author">
15. <input type="submit" name="submit" value="Submit!">
16. </form>
17. <table border="1" cellpadding="3">
18. <?php
19. /* This program gets news items from the database */
20. $db = mysql_connect("localhost", "root");
21. mysql_select_db("php3", $db);
22. if ($orderby == 'date'):
23. $sql = "select * from news order by 'date'";
24. elseif ($orderby == 'author'):
25. $sql = "select * from news order by 'author_name'";
26. elseif ($orderby == 'heading'):
27. $sql = "select * from news order by 'heading'";
28. elseif (isset($submit)):
29. $sql = "select * from news where author_name = '$author'";
30. else:
31. $sql = "select * from news";
32. endif;
33. $result = mysql_query($sql);
34. while ($row = mysql_fetch_array($result)) {
35. print("<tr><td bgcolor=\"#003399\"><b>");
36. printf("<font color=\"white\">%s</font></b></td></tr>\n",
37. $row["heading"]);
38. printf("<td>By: <a href=\"mailto:%s\">%s</a>\n",
```

```
39. $row["author_email"], $row["author_name"]);
40. printf("<br>Posted: %s<hr>\n",
41. $row["date"]);
42. printf("%s</td></tr>\n",
43. $row["body"]);
44. }
45. ?>
46. </table>
47. </body>
48. </html>
```

HOW THE SCRIPT WORKS

- As you remember from previous chapters in the book, these URLs send variables to PHP. The variables in lines 8 through 10 are used by the PHP script to determine which SQL statements it will send to the MySQL server.
- The form in lines 12 through 16 is used to send an optional query to show only those news items written by a single author. No values are returned if the user enters an author name that is not in the database.
- Lines 20 and 21 establish the connection to the MySQL server and select the database that is used for the queries.
- Lines 22 through 32 give the users some power over what kind of information they can get out of the database and how this information is displayed. The links in lines 8 through 10 and the form in lines 12 through 16 determine which of the `if` statements are used. The `orderby` variable drives how the data is displayed. If the `orderby` variable has no value and the Submit button was not pressed, then line 31 is used as the SQL query.
- At line 33, the query must be done and the data is stored in `$result`.
- At line 34, the `mysql_fetch_array` function grabs a row of all the data returned from the query. The `while` statement increments the row after each loop. The loop continues until all the rows returned from the query are processed by the `mysql_fetch_array` function.
- With lines 35 through 43, each of the rows returned by `the mysql_fetch_array` function are printed out using the `printf` function. The columns of each row are referenced by their column name.
- At line 44, the while loop ends after all the rows returned from the SQL query are processed.

◆ Recap

The database lets you modularize your data into convenient chunks of information that are easy to manage, and it makes it easy for the users to get at the information they want.

Database integration with your Web site gives you a lot of flexibility on how you can display information. We could have done the same thing by writing the output of the `data_in` script to a text file. But if you did that, you couldn't selectively display only certain authors, nor could you change the order of the display. Now imagine that there were 1,000 news items to sort through. One large text file containing all that information would make it difficult for users to get at the information that they want.

But the best part is, database integration using PHP is quick and easy. As you saw in Script 7-1, it takes only a few lines of basic code to use a database with PHP.